Function 1. BayesOptRegionQuad

Function `BayesOptRegionQuad` implements Algorithm 1 for full quadratic polynomial response surfaces in two variables. To call it, use command:

```
res <- BayesOptRegionQuad(
  design = design, y = y,
  constr_lb = c(-sqrt(2), -sqrt(2)), constr_ub = c(sqrt(2), sqrt(2)),
  alpha = 0.05, n_post = 500, parallel = TRUE
)
```

where `design` takes the design matrix, `y` takes the observation vector, `constr_lb` and `constr_ub` take the lower and upper bounds of the experimental region, `alpha` specifies the acceptable Type-I error, `n_post` specifies the number of posterior parameter draws, and `parallel` specifies if it utilizes multiple cores to compute for the credible region. The function returns a list. Use command `str(res)` to check its structure:

```
List of 5
 $ optima   :'data.frame': 475 obs. of  2 variables:
  ..$ x1: num [1:475] -0.148 -0.172 -0.337 -0.319 -0.51 ...
  ..$ x2: num [1:475] -0.2289 -0.0685 -0.049 -0.0415 0.0879 ...
 $ opt_hat  :'data.frame': 1 obs. of  2 variables:
  ..$ x1: num -0.232
  ..$ x2: num -0.107
 $ beta_hat : Named num [1:6] 90.058 -1.232 -0.682 -2.44 -2.157 ...
  ..- attr(*, "names")= chr [1:6] "intercept" "x1" "x2" "x1x1" ...
 $ constr_lb: num [1:2] -1.41 -1.41
 $ constr_ub: num [1:2] 1.41 1.41
 - attr(*, "class")= chr "bayescrquad"
```

where `optima` contains all the simulated posterior optima, `opt_hat` contains the point estimate of the optimum, and `beta_hat` contains the point estimate of the polynomial coefficients. Use command:

```
plot(res, xlab = "x1", ylab = "x2")
```

to draw the credible region on the optimum, as shown in Figure C.1. The contours indicate the point estimate of the response surface. The red dot indicates the point estimate of the optimum. The gray convex hull indicates the credible region on the optimum.

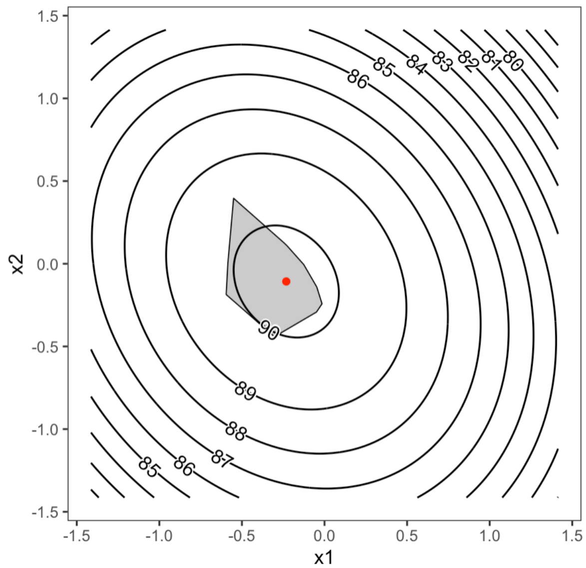

Figure C.1. A credible region generated by `BayesOptRegionQuad`.

Function 2. `BayesOptRegionGP`

Function `BayesOptRegionGP` implements Algorithm 2 for two-variable response surfaces. To call it, use command:

```
c(thetas, thetas_thin, credible_region) %<-% BayesOptRegionGP(
  design = design, y = y,
  constr_lb = c(0, 0), constr_ub = c(5, 5),
  alpha = 0.05, chain_length = 1e5, thin_interval = 200,
  n_path_per_theta = 1, xi = 0.1,
  process_mean_lb = -20, process_mean_ub = 20,
  length_scale_lb = 0, length_scale_ub = 100,
  fun_var_lb = 0, fun_var_ub = 60,
  noise_var_lb = 0, noise_var_ub = 2, parallel = TRUE
)
```

where `chain_length` specifies the length of the stabilized Markov chain, `thin_interval` specifies how often to take a sample from the chain to thin it, `n_path_per_theta` specifies how many sample paths to simulate based on each posterior draw of the GP parameters, `xi` specifies the penalization parameter in Algorithm 3, and `process_mean_lb` ... `noise_var_ub` specify the lower and upper bounds for the prior distributions. The function returns three lists, `thetas`, `thetas_thin`, and `credible_region`. List `thetas` contains the information of the full Markov chain. Use command `str(thetas)` to check its structure:

```
List of 4
 $ process_mean: num [1:100000] -0.18 0.374 -0.299 1.279 2.26 ...
 $ length_scale: num [1:100000] 87.4 81.3 75.7 69.8 75.8 ...
 $ fun_var     : num [1:100000] 3.2 2.36 2.39 1.76 2.24 ...
 $ noise_var   : num [1:100000] 0.00189 0.00325 0.00167 0.00185 0.00136 ...
 - attr(*, "class")= chr "mcmcdraw"
 - attr(*, "row.names")= int [1:100000] 1 2 3 4 5 6 7 8 9 10 ...
```

use command `summary(thetas)` to check selected percentiles of the posterior distributions:

```
                     2.5%           25%          50%          75%       97.5%
process_mean -2.919125e+00 -0.9575027786 -0.089202849  0.743059115  2.69359309
length_scale  1.354765e+01 48.3615793571 69.788245172 86.065808638 98.72522380
fun_var       3.768294e-01  1.3482097334  1.997832656  2.651731082  4.05351362
noise_var     6.544189e-05  0.0007788914  0.001954213  0.004096572  0.01300925
```

and use command `plot(thetas)` to generated the trace and density plots of the the posterior distributions, as shown in Figure C.2. List `thetas_thin` contains the information of the thinned Markov chain and can be explored in the same way. List `credible_region` contains the information of the credible region, use command `str(credible_region)` to check its structure:

```
List of 5
 $ optima   :'data.frame': 475 obs. of  2 variables:
  ..$ x1: num [1:475] 1.12 1.07 1.12 1.12 1.07 ...
  ..$ x2: num [1:475] 1.06 1.43 1.06 1.06 1.43 ...
 $ opt_hat  :'data.frame': 1 obs. of  2 variables:
  ..$ x1: num 1.12
  ..$ x2: num 1.06
 $ rs_hat   :function (x)
  ..- attr(*, "srcref")= 'srcref' int [1:8] 139 3 145 3 3 3 139 145
  .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile'
          <environment: 0x7fd026527b18>
 $ constr_lb: num [1:2] 0 0
 $ constr_ub: num [1:2] 5 5
 - attr(*, "class")= chr "bayescrgp"
```

where `rs_hat` contains the point estimate of the response surface. Use command:

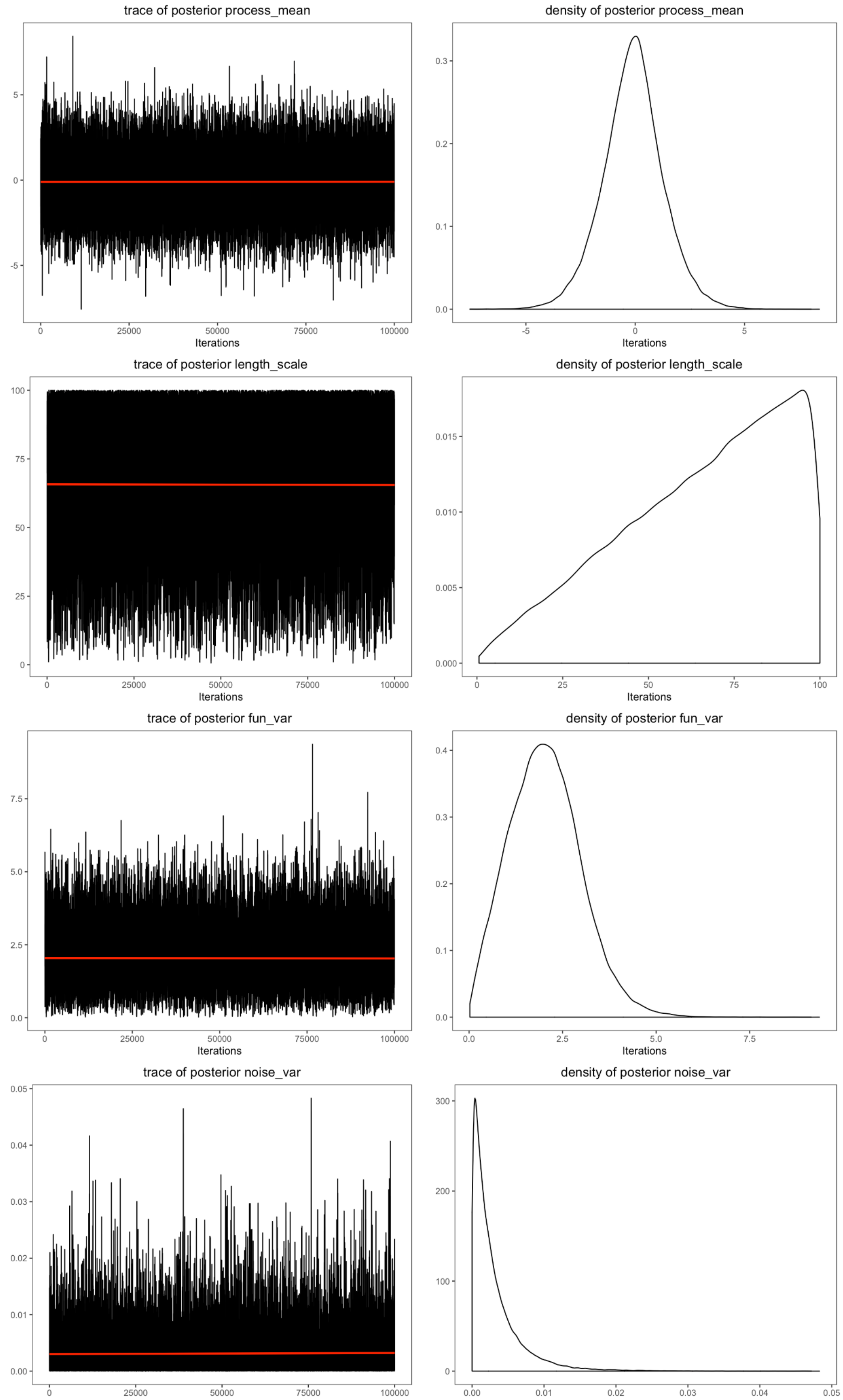

Figure C.2. Trace plots and posterior densities generated by `BayesOptRegionGP`.

```
plot(credible_region, xlab = "x1", ylab = "x2")
```

to draw the credible region on the optimum, as shown in Figure C.3.

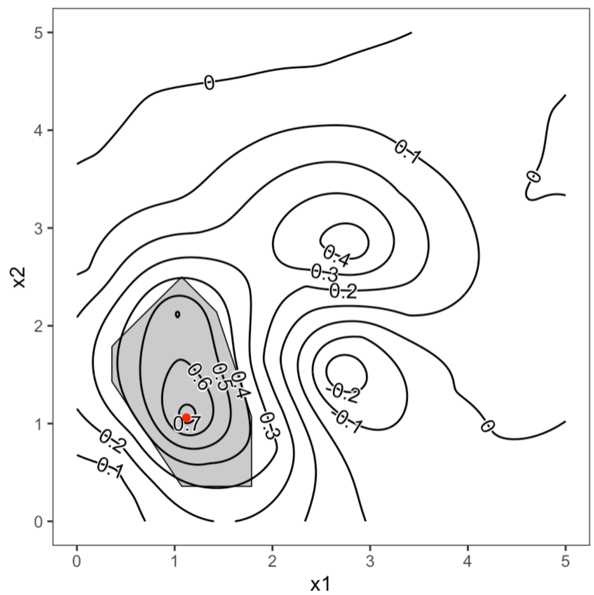

Figure C.3. A credible region generated by `BayesOptRegionGP`.

Function 3. `OptRegionQuad`

Function `OptRegionQuad` implements Algorithm 5 for full quadratic polynomial response surfaces in two variables. To call it, use command:

```
res <- OptRegionQuad(
  design = design, y = y,
  constr_lb = c(-sqrt(2), -sqrt(2)), constr_ub = c(sqrt(2), sqrt(2)),
  alpha = 0.05, B = 200
)
```

where `B` specifies the size of the bootstrap. The function returns a list. Use command `str(res)` to check its structure:

```
List of 6
 $ optima   :'data.frame': 190 obs. of  2 variables:
  ..$ X1: num [1:190] -0.171 -0.299 -0.359 -0.236 -0.165 ...
  ..$ X2: num [1:190] -0.207 -0.0555 -0.1021 -0.0106 -0.2206 ...
```

```
$ opt_bag  :'data.frame': 1 obs. of  2 variables:
  ..$ X1: num -0.229
  ..$ X2: num -0.11
$ design   :'data.frame': 22 obs. of  2 variables:
  ..$ X1: num [1:22] -1 1 -1 1 -1.41 ...
  ..$ X2: num [1:22] -1 -1 1 1 0 ...
$ y        : num [1:22, 1] 87.6 86 87.3 83.3 86.9 ...
$ constr_lb: num [1:2] -1.41 -1.41
$ constr_ub: num [1:2] 1.41 1.41
- attr(*, "class")= chr "crquad"
```

where `opt_bag` contains the bootstrap aggregated optimum. Use command:

```
plot(res, xlab = "x1", ylab = "x2")
```

to draw the confidence region on the optimum, as shown in Figure C.4. The red dots

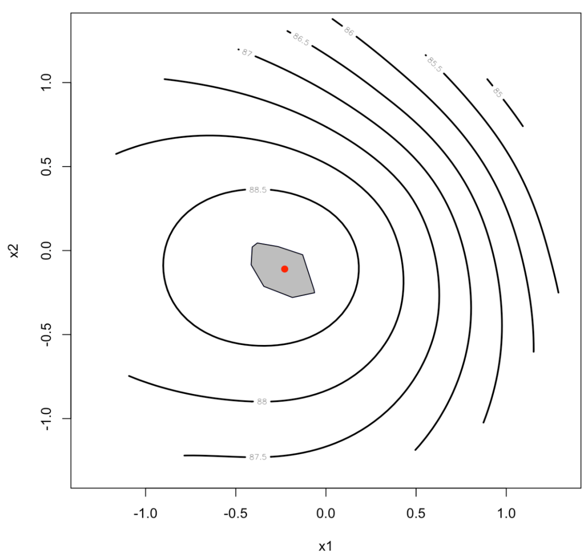

Figure C.4. A confidence region generated by `OptRegionQuad`.

indicates the bootstrap aggregated optimum.

Function 4. `GloptiPolyRegion`

Function `GloptiPolyRegion` implements Algorithm 5 for polynomial response surfaces up to cubic order in $3 \sim 5$ variables. To call it, use command:

```
res <- GloptiPolyRegion(
  design = cubic_5D$design_matrix, y = cubic_5D$response,
  constr_lb = rep(0, 5), constr_ub = rep(5, 5),
  alpha = 0.05, B = 1000, degree = 3
)
```

where `degree` specifies the order of the polynomial model. The function returns a list. Use command `str(res)` to check its structure:

```
List of 4
 $ optima   :'data.frame': 950 obs. of  5 variables:
  ..$ X1: num [1:950] 5 2 5 2.76 3.05 ...
  ..$ X2: num [1:950] 2.42 2.13 2.26 2.34 2.36 ...
  ..$ X3: num [1:950] 0.858 1.16 1.084 1.096 1.06 ...
  ..$ X4: num [1:950] 2.69 2.57 2.37 2.53 2.68 ...
  ..$ X5: num [1:950] 2.25 2.51 2.28 2.73 2.47 ...
 $ opt_bag  :'data.frame': 1 obs. of  5 variables:
  ..$ X1: num 3.81
  ..$ X2: num 2.37
  ..$ X3: num 1.01
  ..$ X4: num 2.62
  ..$ X5: num 2.49
 $ constr_lb: num [1:5] 0 0 0 0 0
 $ constr_ub: num [1:5] 5 5 5 5 5
 - attr(*, "class")= chr "crpoly"
```

Use command:

```
plot(res, axes_labels = c("x1", "x2", "x3", "x4", "x5"))
```

to draw pairwise projections of the confidence confidence on the optimum, as shown in Figure C.5

Function 5. `OptRegionTps`

Function `OptRegionTps` implements Algorithm 6 for two-variable response surfaces. To call it, use command:

```
res <- OptRegionTps(
```

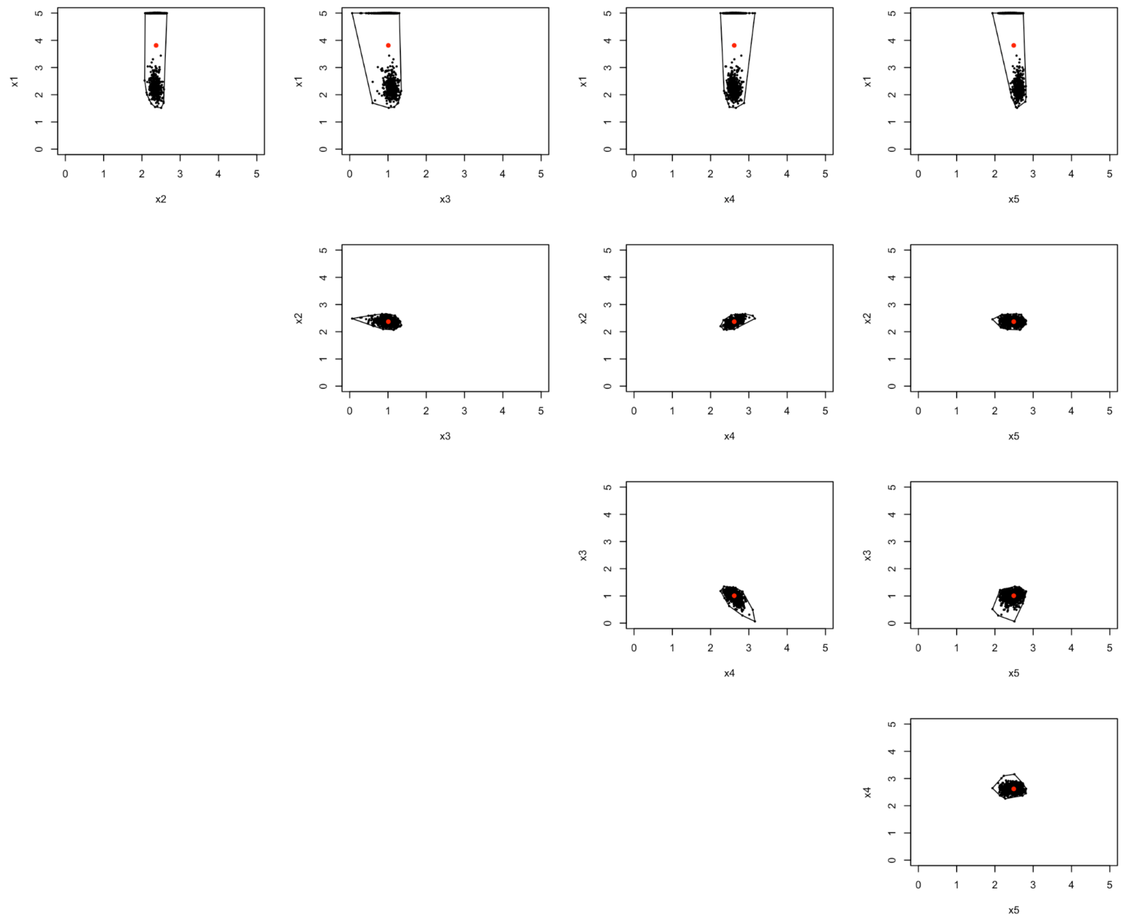

Figure C.5. Projections of a confidence region generated by `GloptiPolyRegion`.

```
  design = design, y = y,
  constr_lb = c(0, 0), constr_ub = c(5, 5),
  alpha = 0.05, B = 200
)
```

The function returns a list. Use command `str(res)` to check its structure:

```
List of 7
 $ optima   :'data.frame': 190 obs. of  2 variables:
  ..$ X1: num [1:190] 0.928 0.75 0.991 0.704 0.889 ...
  ..$ X2: num [1:190] 1.56 1.82 1.82 1.62 1.51 ...
 $ opt_bag  :'data.frame': 1 obs. of  2 variables:
  ..$ X1: num 0.866
  ..$ X2: num 1.69
 $ design   :'data.frame': 200 obs. of  2 variables:
```

```
  ..$ X1: num [1:200] 2.21 1.579 2.829 3.85 0.236 ...
  ..$ X2: num [1:200] 3.04 2.24 4.34 2.95 4.44 ...
 $ y        : num [1:200, 1] 0.4668 0.3331 0.0693 0.079 -0.0113 ...
 $ lambda   : num 0.04
 $ constr_lb: num [1:2] 0 0
 $ constr_ub: num [1:2] 5 5
 - attr(*, "class")= chr "crtps"
```

where `lambda` is the penalization parameter value used to fit Thin Plate Spline model. Use commmand:

```
plot(res, xlab = "x1", ylab = "x2")
```

to draw the confidence region on the optimum, as shown in Figure C.6.

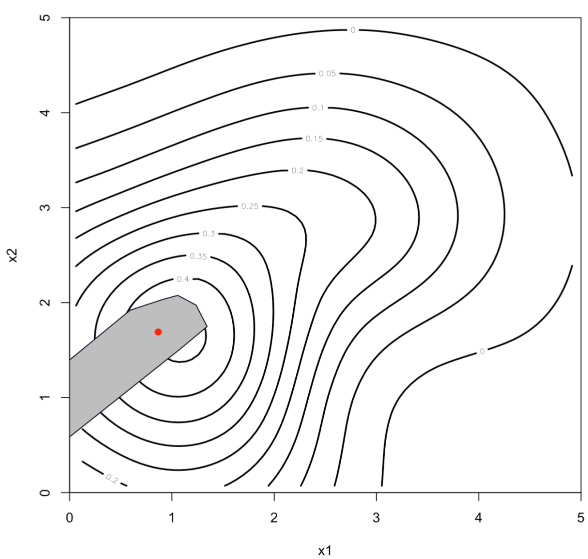

Figure C.6. A confidence region generated by `OptRegionTps`.

Function 6. `OptRegionGP`

Function `OptRegionGP` implements Algorithm 7 for two-variable response surfaces. To call it, use command:

```
res <- OptRegionGP(
  design = design, y = y,
  constr_lb = c(0, 0), constr_ub = c(5, 5),
```

```
  alpha = 0.05, B = 1000, xi = 0.1, parallel = TRUE
)
```

The function returns a list. Use command `str(res)` to check its structure:

```
List of 5
 $ optima   :'data.frame': 950 obs. of  2 variables:
  ..$ x1: num [1:950] 1.43 1.43 1.43 1.43 1.43 ...
  ..$ x2: num [1:950] 1.79 1.79 1.79 1.79 1.79 ...
 $ opt_bag  :'data.frame': 1 obs. of  2 variables:
  ..$ x1: num 1.11
  ..$ x2: num 1.81
 $ rs_hat   :function (x)
  ..- attr(*, "srcref")= 'srcref' int [1:8] 214 3 220 3 3 3 214 220
  .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile'
         <environment: 0x7f963bfe5488>
 $ constr_lb: num [1:2] 0 0
 $ constr_ub: num [1:2] 5 5
 - attr(*, "class")= chr "crgpok"
```

Use commmand:

```
plot(res, xlab = "x1", ylab = "x2")
```

to draw the confidence region on the optimum, as shown in Figure C.7.

Function 7. `OptRegionOK`

Function `OptRegionOK` implements Algorithm 9 for two-variable response surfaces. To call it, use command:

```
res <- OptRegionOK(
  design = design, y = y,
  constr_lb = c(0, 0), constr_ub = c(5, 5),
  alpha = 0.05, B = 1000, xi = 0.1, parallel = TRUE
)
```

The function returns a list. Use command `str(res)` to check its structure:

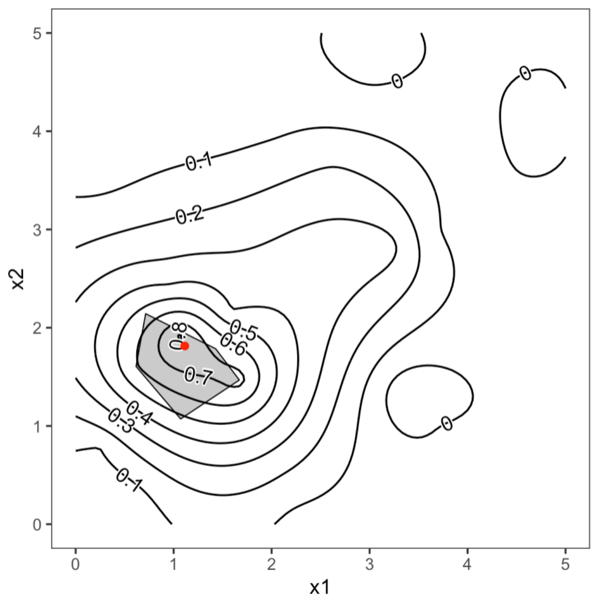

Figure C.7. A confidence region generated by `OptRegionGP`.

```
List of 5
 $ optima   :'data.frame': 950 obs. of  2 variables:
  ..$ x1: num [1:950] 1.06 1.06 1.06 1.06 1.06 ...
  ..$ x2: num [1:950] 1.91 1.91 1.91 1.91 1.91 ...
 $ opt_bag  :'data.frame': 1 obs. of  2 variables:
  ..$ x1: num 1.13
  ..$ x2: num 1.83
 $ rs_hat   :function (x)
  ..- attr(*, "srcref")= 'srcref' int [1:8] 214 3 220 3 3 3 214 220
  .. ..- attr(*, "srcfile")=Classes 'srcfilecopy', 'srcfile'
        <environment: 0x7f963bfe5488>
 $ constr_lb: num [1:2] 0 0
 $ constr_ub: num [1:2] 5 5
 - attr(*, "class")= chr "crgpok"
```

Use commmand:

```
plot(res, xlab = "x1", ylab = "x2")
```

to draw the confidence region on the optimum, as shown in Figure C.8.

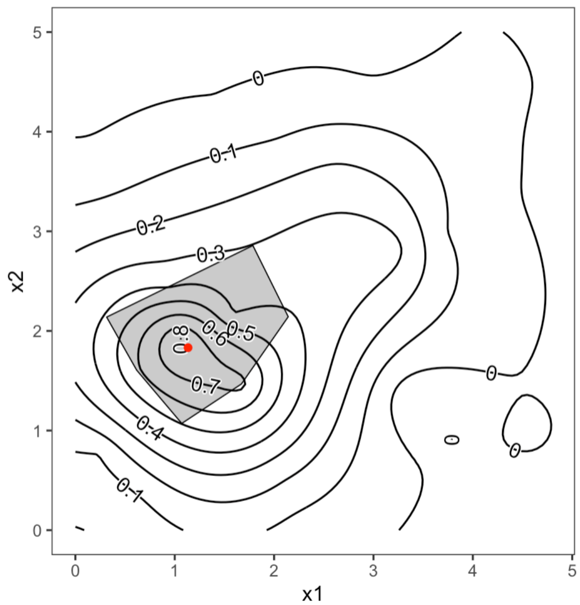

Figure C.8. A confidence region generated by `OptRegionOK`.

Function 8. `OptRegionPC1Quad`

Function `OptRegionPC1Quad` takes high-dimensional observations in two controllable factors and compute a confidence region on the optimum of the assumed quadratic probabilistic first principal response surface. To call it, use command:

```
c(cr, ppca) %<-% OptRegionPC1Quad(
  design = design, Y = Y,
  constr_lb = c(-sqrt(2), -sqrt(2)), constr_ub = c(sqrt(2), sqrt(2)),
  alpha = 0.05, B = 1000
)
```

where Y takes the observation matrix. The function returns two lists, `cr` and `ppca`. List `ppca` contains the information of the fitted PPCA model. Use command `str(ppca)` to check its structure:

```
List of 5
 $ mu_hat    : num [1:30] 4.84 -1.73 -2.52 1.88 2.54 ...
 $ sigma2_hat: num 86.3
 $ W_hat     : num [1:30, 1] 8.329 -0.0943 0.4814 3.9147 0.5941 ...
 $ z_given_x : num [1, 1:22] -0.1061 -0.8549 -0.0923 0.0157 -0.9758 ...
 $ props     : num [1:30] 0.1451 0.1263 0.1039 0.0886 0.0878 ...
```

where mu_hat contains the fitted high-dimensional mean vector, sigma2_hat contains the fitted noise variance , W_hat contains the fitted dimension-transformation matrix, z_given_x contains the probabilistic first principal responses at the design points, and props contains the standard PCA component variance proportions. List cr contains the information of the confidence region. Use command str(cr) to check its structure:

```
List of 6
 $ optima   :'data.frame': 950 obs. of  2 variables:
  ..$ X1: num [1:950] -0.0805 -0.0161 -0.039 -0.129 0.067 ...
  ..$ X2: num [1:950] -0.3 -0.171 -0.167 -0.362 -0.178 ...
 $ opt_bag  :'data.frame': 1 obs. of  2 variables:
  ..$ X1: num -0.0451
  ..$ X2: num -0.153
 $ design   :'data.frame': 22 obs. of  2 variables:
  ..$ X1: num [1:22] -1 1 -1 1 -1.41 ...
  ..$ X2: num [1:22] -1 -1 1 1 0 ...
 $ y        : num [1:22] -0.1061 -0.8549 -0.0923 0.0157 -0.9758 ...
 $ constr_lb: num [1:2] -1.41 -1.41
 $ constr_ub: num [1:2] 1.41 1.41
 - attr(*, "class")= chr "crquad"
```

Use command:

```
plot(cr, xlab = "x1", ylab = "x2")
```

to draw the confidence region on the optimum, as shown in Figure C.9.

Function 9. GloptipolyR

Function GloptiPolyR implements the Gloptipoly (Lasserre, 2001) algorithm. Consider optimizing the following quadratic function in three variables:

$$f(\boldsymbol{x}) = -1.5x_1 + 2.13x_2 - 1.81x_3 + 7.13x_1x_2 + 3.27x_1x_3 + 2.73x_2x_3 + 4.69x_1^2 + 6.27x_2^2 + 5.21x_3^2$$

defined over $\mathcal{X} = \{-2 \leq x_i \leq 2, i = 1, 2, 3\}$, with its global minimum at $\boldsymbol{x}^* =$ (0.46, -0.46, 0.15). The optimization problem can be formally written as:

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) \\ \text{subject to:} \quad & g_1(\boldsymbol{x}) = x_1 + 2 \geq 0 \end{aligned}$$

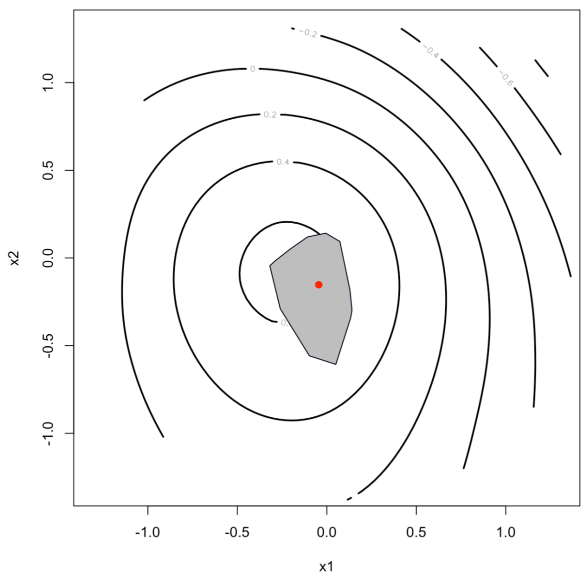

Figure C.9. A confidence region generated by `OptRegionPC1Quad`.

$$g_2(\boldsymbol{x}) = x_1 - 2 \leq 0$$
$$g_3(\boldsymbol{x}) = x_2 + 2 \geq 0$$
$$g_4(\boldsymbol{x}) = x_2 - 2 \leq 0$$
$$g_5(\boldsymbol{x}) = x_3 + 2 \geq 0$$
$$g_6(\boldsymbol{x}) = x_3 - 2 \leq 0$$

The input for `GloptiPolyR` is a list `P` of seven sub-lists, corresponding to $f(\boldsymbol{x})$, $g_1(\boldsymbol{x})$, $g_2(\boldsymbol{x})$, $\cdots$, $g_6(\boldsymbol{x})$, respectively:

```
P <- list()
p_f <- list()
p_g_1 <- list(); p_g_2 <- list(); p_g_3 <- list()
p_g_4 <- list(); p_g_5 <- list(); p_g_6 <- list()
```

Each of these seven sub-lists has two elements: (1) a multi-dimensional array, denoted by 'c', and (2) an attribute, denoted by 't'. The multi-dimensional array is generated from the monomial coefficients of the corresponding polynomial function. The rule is to put the coefficient of the $x_1^i x_2^j x_3^k$ term in the $[i+1, j+1, k+1]$ position of the array, and place zeroes in other positions:

```
p_f$c <- array(0, dim = c(3, 3, 3))
p_f$c[2, 1, 1] <- -1.5; p_f$c[1, 2, 1] <- 2.13; p_f$c[1, 1, 2] <- -1.81
```

```
p_f$c[2, 2, 1] <- 7.13; p_f$c[2, 1, 2] <- 3.27; p_f$c[1, 2, 2] <- 2.73
p_f$c[3, 1, 1] <- 4.69; p_f$c[1, 3, 1] <- 6.27; p_f$c[1, 1, 3] <- 5.21

p_g_1$c <- array(0, dim = c(3, 3, 3))
p_g_1$c[1, 1, 1] <- 2; p_g_1$c[2, 1, 1] <- 1

p_g_2$c <- array(0, dim = c(3, 3, 3))
p_g_2$c[1, 1, 1] <- -2; p_g_2$c[2, 1, 1] <- 1

p_g_3$c <- array(0, dim = c(3, 3, 3))
p_g_3$c[1, 1, 1] <- 2; p_g_3$c[1, 2, 1] <- 1

p_g_4$c <- array(0, dim = c(3, 3, 3))
p_g_4$c[1, 1, 1] <- -2; p_g_4$c[1, 2, 1] <- 1

p_g_5$c <- array(0, dim = c(3, 3, 3))
p_g_5$c[1, 1, 1] <- 2; p_g_5$c[1, 1, 2] <- 1

p_g_6$c <- array(0, dim = c(3, 3, 3))
p_g_6$c[1, 1, 1] <- -2; p_g_6$c[1, 1, 2] <- 1
```

Next, set the attribute for the objective function as either “min” or “max”:

```
p_f$t <- "min"
```

Then, set the attributes for the constraint functions as either “>=” or “<=”:

```
p_g_1$t <- ">="; p_g_2$t <- "<="
p_g_3$t <- ">="; p_g_4$t <- "<="
p_g_5$t <- ">="; p_g_6$t <- "<="
```

Finally, we construct the list `P` from the 7 sub-lists and use it to call `GloptiPolyR`:

```
P <- list(p_f, p_g_1, p_g_2, p_g_3, p_g_4, p_g_5, p_g_6)
res <- GloptiPolyR(P)
```

then, use command `str(res)` to retrieve the optimization result:

```
List of 2
 $ solution : num [1:3] 0.46 -0.465 0.151
 $ objective: num -0.977
```